Лабораторная работа №5

В проекте были применены следующие паттерны проектирования:

1) Паттерн проектирования «Singleton»

Используем единственный экземпляр класса для того, чтобы создать объекты в любом месте программы. Используется для подключения к базе данных.

```
private static GameObjectCreator instance;

public static GameObjectCreator getInstance() {
    if (instance == null) {
        instance = new GameObjectCreator();
    }
    return instance;
}
```

```
private static SQLiteConnection instance;

public static SQLiteConnection getInstance() {
    if (instance == null) {
        instance = new SQLiteConnection();
    }

    return instance;
}
```

2) Паттерн проектирования «Наблюдатель»

Наблюдатель слушает все ходы и делает все необходимые действия на карте и в коллекции.

```
public interface MoveResultNotifier {

    List<MoveResultListener> getMoveListeners();

    void addMoveListener(MoveResultListener listener);

    public void removeMoveListener(MoveResultListener listener);

    public void removeAllMoveListeners();

    public void notifyMoveListeners(ActionResult actionResult,
AbstractGameObject movingObject, AbstractGameObject targetObject);

}
```

3) Паттерн проектирования «Фасад»

Это структурный паттерн проектирования, который предоставляет простой интерфейс к сложной системе классов, библиотеке или фреймворку. Фасад — это простой интерфейс для работы со сложной подсистемой, содержащей множество классов. Фасад может иметь урезанный интерфейс, не имеющий 100% функциональности, которой можно достичь, используя сложную подсистему напрямую. Но он предоставляет именно те фичи, которые нужны клиенту, и скрывает все остальные.

```
    public class GameFacade {

    private HybridMapLoader mapLoader;
    private SoundPlayer soundPlayer;
    private ScoreSaver scoreSaver;
    private MapInfo mapInfo;
    private AbstractGameMap gameMap;

    public GameFacade(HybridMapLoader mapLoader, SoundPlayer soundPlayer,
ScoreSaver scoreSaver) {
        this.mapLoader = mapLoader;
        this.scoreSaver = scoreSaver;
        this.soundPlayer = soundPlayer;
    }

    public GameFacade() {
    }

    public void setSoundPlayer(SoundPlayer soundPlayer) {
        this.soundPlayer = soundPlayer;
    }

    public void setMapLoader(HybridMapLoader mapLoader) {
        this.mapLoader = mapLoader;

        // слушатели для звуков должны идти в первую очередь, т.к. они
запускаются в отдельном потоке и не мешают выполняться следующим слушателям
        if (soundPlayer instanceof MoveResultListener) {

mapLoader.getGameMap().getGameCollection().addMoveListener((MoveResultListene
r) soundPlayer); // реализация Паттерна Наблюдатель
        }

        updateMap();
    }

    public ScoreSaver getScoreSaver() {
        return scoreSaver;
    }

    public void setScoreSaver(ScoreSaver scoreSaver) {
        this.scoreSaver = scoreSaver;
    }

    public void stopGame() {
```

```
        soundPlayer.stopBackgoundMusic();
        mapLoader.getGameMap().stop();
    }

    public void moveObject(MovingDirection movingDirection, GameObjectType
gameObjectType) {
        gameMap.getGameCollection().moveObject(movingDirection,
gameObjectType);
    }

    public Component getMap() {
        return mapLoader.getGameMap().getMapComponent();
    }

    public void saveScore() {
        UserScore userScore = new UserScore();
        userScore.setUser(mapInfo.getUser());
        userScore.setScore(getGoldMan().getTotalScore());
        scoreSaver.saveScore(userScore);
    }

    public void addMoveListener(MoveResultListener listener) {
        mapLoader.getGameMap().getGameCollection().addMoveListener(listener);
// реализация Паттерна Наблюдатель
    }

    public void saveMap() {
        SavedMapInfo saveMapInfo = new SavedMapInfo();
        saveMapInfo.setId(mapInfo.getId());
        saveMapInfo.setUser(mapInfo.getUser());
        saveMapInfo.setTotalScore(getGoldMan().getTotalScore());
        saveMapInfo.setTurnsCount(getGoldMan().getTurnsNumber());
        mapLoader.saveMap(saveMapInfo, LocationType.DB);
    }

    public void startGame() {
        soundPlayer.startBackgroundMusic(WavPlayer.SOUND_BACKGROUND);
        mapLoader.getGameMap().start();
    }

    private GoldMan getGoldMan() {
        return (GoldMan)
mapLoader.getGameMap().getGameCollection().getGameObjects(GameObjectType.GOLD
MAN).get(0);
    }

    public int getTurnsLeftCount() {
        return mapInfo.getTurnsLimit() - getGoldMan().getTurnsNumber();
    }

    public int getTotalScore() {
        return getGoldMan().getTotalScore();
    }

    public void updateMap() {
        gameMap = mapLoader.getGameMap();
        mapInfo = gameMap.getMapInfo();


        gameMap.updateMap();
    }

    public void updateObjects(AbstractGameObject obj1, AbstractGameObject
obj2) {
```

```
        gameMap.updateMapObjects(obj1, obj2);
    }
}
```

4) Паттерн проектирования «Адаптер».

Адаптер — это структурный паттерн проектирования, который позволяет объектам с несовместимыми интерфейсами работать вместе.

```
public class HybridMapLoader {

    private DBMapLoader dBMapLoader;
    private FSMapLoader fSMapLoader;

    private AbstractGameMap gameMap;

    public HybridMapLoader(AbstractGameMap gameMap) {
        dBMapLoader = new DBMapLoader(gameMap);
        fSMapLoader = new FSMapLoader(gameMap);
        this.gameMap = gameMap;
    }



    public boolean saveMap(SavedMapInfo mapInfo, LocationType locationType){
        switch (locationType){
            case DB:{
                return dBMapLoader.saveMap(mapInfo);
            }

            case FS:{
                return fSMapLoader.saveMap(mapInfo);
            }
        }

        return false;
    }

    public boolean loadMap(MapInfo mapInfo, LocationType locationType){
        switch (locationType){
            case DB:{
                gameMap = dBMapLoader.getGameMap();
                return dBMapLoader.loadMap(mapInfo);
            }

            case FS:{
                gameMap = fSMapLoader.getGameMap();
                return fSMapLoader.loadMap(mapInfo);
            }
        }



        return false;
    }

    public ArrayList<SavedMapInfo> getSavedMapList(User user, LocationType
locationType){
        switch (locationType){
            case DB:{
```

```
                return dBMapLoader.getSavedMapList(user);
            }

            case FS:{
                return fSMapLoader.getSavedMapList(user);
            }
        }

        return null;
    }

    public boolean deleteSavedMap(MapInfo mapInfo, LocationType
locationType){
        switch (locationType){
            case DB:{
                return dBMapLoader.deleteSavedMap(mapInfo);
            }

            case FS:{
                return fSMapLoader.deleteSavedMap(mapInfo);
            }
        }

        return false;
    }

    public AbstractGameMap getGameMap() {
        return gameMap;
    }


    public int getPlayerId(String username){
        return dBMapLoader.getPlayerId(username);
    }

}
```